

NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



THESIS

HYPERMEDIA PROCESS KNOWLEDGE MAPPING (HYPERPKM)

by

Christopher L. Vance

and

Kevin P. Sudhoff

September 1995

Principal Advisor:

Balasubramaniam Ramesh

Approved for public release; distribution is unlimited.

19960215 012

DTIC QUALITY INSPECTED 3

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 1995		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE HYPERMEDIA PROCESS KNOWLEDGE MAPPING (HYPERPKM)			5. FUNDING NUMBERS	
6. AUTHOR(S) Vance, Christopher L. Sudhoff, Kevin P.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The capture and reuse of design decisions and their rationale is increasingly being recognized as very important in increasing software development and maintenance productivity. By using these concepts, the DOD has recognized the ability to achieve a significant reduction in development and maintenance costs on all software development projects. The REMAP model provides the primitives and mechanisms for a structured representation of this information. Our thesis implements a graphical user interface for the REMAP model to facilitate easy acquisition and reuse of process knowledge. Much of the "informal" components of design decision and rationale may be represented using multimedia documents. The ability to link such documents and search for "relevant" components of process knowledge from these is a key attribute of the HyperPKM model. HyperPKM provides the capability to link and search multimedia documents distributed in the WWW to the REMAP objects displayed in the graph browser.				
14. SUBJECT TERMS REMAP, HyperPKM, Hypermedia, Indexing, Search, Retrieval			15. NUMBER OF PAGES 60	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Approved for public release; distribution is unlimited.

HYPERMEDIA PROCESS KNOWLEDGE MAPPING (HYPERPKM)

Christopher L. Vance
Lieutenant Commander, United States Navy
B.S., U.S. Naval Academy, 1982

Kevin P. Sudhoff
Lieutenant, United States Navy
B.S., Auburn University, 1988

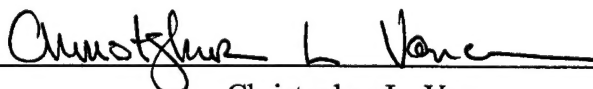
Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 1995

Authors:

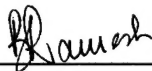


Christopher L. Vance

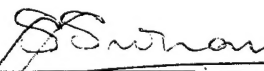


Kevin P. Sudhoff


Approved by:



Balasubramaniam Ramesh, Principal Advisor



Suresh Sridhar, Associate Advisor



Reuben T. Harris, Chairman, Department of Systems Management

ABSTRACT

The capture and reuse of design decisions and their rationale is increasingly being recognized as very important in increasing software development and maintenance productivity. By using these concepts, the DOD has recognized the ability to achieve a significant reduction in development and maintenance costs on all software development projects. The REMAP model provides the primitives and mechanisms for a structured representation of this information. Our thesis implements a graphical user interface for the REMAP model to facilitate easy acquisition and reuse of process knowledge. Much of the “informal” components of design decision and rationale may be represented using multimedia documents. The ability to link such documents and search for “relevant” components of process knowledge from these is a key attribute of the HyperPKM model. HyperPKM provides the capability to link and search multimedia documents distributed in the WWW to the REMAP objects displayed in the graph browser.

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	GENERAL	1
B.	THESIS OBJECTIVES	2
C.	APPLICABILITY TO THE DEPARTMENT OF DEFENSE	2
D.	SCOPE AND PREPARATION	2
E.	ORGANIZATION OF THE STUDY	3
II.	REMAP PROJECT ENVIRONMENT	5
A.	REMAP BACKGROUND	5
B.	REMAP CONCEPTBASE IMPLEMENTATION	6
III.	EXTENDING THE REMAP GRAPHBROWSER UTILITY (REMAP GBU) PORTION OF HYPERPKM	9
A.	INTRODUCTION	9
B.	ANDREW TOOLKIT (ATK) OVERVIEW	9
1.	Introduction	9
2.	Object Oriented Programming Environment	9
C.	REMAP MODULES AND CLASS STRUCTURES	10
1.	Rmain Module	10
2.	Modul Header	10
3.	Rapp Class	10
4.	RCB Class	11

5.	Rgraphv Graphview Class	11
6.	Robject Class	11
7.	Requirement Class	11
8.	Issue Class	12
9.	Position Class	12
10.	Sellist Selectionlist Class	12
D.	REVISION CONTROL	12
1.	Purpose of Revision Control	12
2.	The Revision Control System (RCS)	12
3.	Managing REMAP GBU Revisions Using RCS	13
E.	CODING HYPERPKM (REMAP GBU EXTENSION)	13
1.	Creating the Position Class	13
2.	Creating the <i>Search Related Documents</i> Class Method	18
F.	COMPILATION OF HYPER PKM	19
1.	Setting Environment Variables	19
2.	Creating the Makefile	19
3.	Makefile Generation and Code Compilation	20
IV.	SEARCHING VIA THE WORLD-WIDE WEB	21
A.	EXISTING SEARCH OPTIONS	21
B.	INFORMATION INDEXING AND SEARCHING	22
C.	IMPROVEMENTS TO KEYWORD SEARCH ENGINES	23
V.	OVERVIEW OF THE HYPERPKM INDEXING GATEWAY	25

A.	SITE INDEXING	25
1.	Stop Wording	25
2.	Word Frequency	26
3.	Abbreviations and Acronyms	26
4.	Word Stemming	26
5.	Boolean Searches	27
B.	THESAURUS	27
C.	WORLD-WIDE WEB INTERFACE	28
VI.	EXAMPLE OF THE HYPERPKM INDEXING GATEWAY	31
VII.	RECOMMENDATIONS AND CONCLUSIONS	45
A.	DEVELOPER REQUIREMENTS	45
B.	HELPFUL REFERENCES	45
C.	RECOMMENDATIONS	45
D.	FUTURE WORK	46
	LIST OF REFERENCES	47
	INITIAL DISTRIBUTION LIST	49

LIST OF FIGURES

Figure 1.	REMAP Conceptual Model	7
Figure 2.	REMAP GBU Session	32
Figure 3.	HyperPKM Indexing Gateway	34
Figure 4.	Searchable Document Hierarchy	35
Figure 5.	Keyword and Selected Hierarchy Data Field	36
Figure 6.	Document Resulting From Search Query	37
Figure 7.	Alternative Search Technique	38
Figure 8.	Output From Alternative Query	39
Figure 9.	Selection of the Thesaurus Search Feature	40
Figure 10.	Search With Thesaurus Function Enabled	42
Figure 11.	Thesaurus Search By Topic Area	43
Figure 12.	Document Retrieved Via Hypertext Link	44

I. INTRODUCTION

A. GENERAL

This thesis describes the development and implementation of the initial prototype of Hypermedia Process Knowledge Mapping (HyperPKM). The HyperPKM model is a structured methodology by which stakeholders of software design projects can achieve improved documentation of design rationale and decisions. This improved documentation is performed by attaching (or linking) hypermedia documents to the objects of a structured conceptual model (REMAP), which captures design rationale and software design process knowledge. The implementation of this prototype of HyperPKM is the culmination of an intensive development effort which includes the extension of the REMAP GBU and incorporation of a hypermedia search engine for use via the World-Wide Web (WWW--<http://sm.nps.navy.mil/webmaster/guide61/guide.01.html>).

One of the major themes and driving forces of the use of hypermedia applications, is its ability to effectively and expressively communicate information to the user of the application [Ref. 1]. These complex hypermedia applications are typically very costly to build, difficult to debug, inconsistent, or very slow to run, depending upon the tools with which they are built. The Andrew Toolkit (ATK) helps the programmer avoid many of the pitfalls and difficult problems in building such applications, and includes numerous helpful tools for building hypermedia interfaces [Ref. 2]. ATK is a C language based utility program which uses dynamic linking, preprocessing, and an inheritance mechanism to make this complex programming task easier. The ATK and hypermedia application development will be discussed in Chapter III.

Since ATK includes a vast hypermedia applications development toolkit, it is used as the basis of the implementation of the Representation and Maintenance of Process knowledge (REMAP) Graph Browser Utility (GBU). REMAP GBU provides support for the various stakeholders in software development projects by permitting the structured capture of the history of design decisions and rationale during the early stages of the project

life cycle [Ref. 3]. The REMAP model, GBU and its benefits will be more thoroughly discussed in Chapter II.

B. THESIS OBJECTIVES

A primary objective of this research is to develop a mechanism to link hypermedia documents representing process knowledge to instances of REMAP objects. The REMAP model, which is at the heart of HyperPKM, is implemented in the ConceptBase, a knowledge base management system. The implementation of REMAP within HyperPKM includes a hypermedia search engine, and mechanism to link internal (concept base) and external (WWW) hypermedia search query results to the application.

C. APPLICABILITY TO THE DEPARTMENT OF DEFENSE

There is a drastic need within the Department of Defense (DOD) to significantly reduce development and maintenance costs on all software development projects. HyperPKM provides a vehicle by which large scale software development and maintenance projects can be more economically and efficiently managed. Additionally, the HyperPKM model is easily ported across a wide spectrum of workstations, enabling its use as a standard software development tool throughout DOD.

D. SCOPE AND PREPARATION

The scope of this thesis is limited to a detailed review of hypermedia concepts, the REMAP model and hypermedia search engine capabilities. The remainder is a description of the design and implementation of HyperPKM to include the incorporation of the Position Object, the WWW hypermedia documents search engine and the use of hypermedia to capture and maintain process knowledge.

Advanced knowledge of the PERL scripting language and the C programming language, as well as a basic understanding of the Andrew Toolkit are required for this work. Preparatory work included an intensive C programming course at the University of California at Santa Cruz, international cooperation in learning and utilizing the PERL scripting language for use with the HyperPKM model, an extensive e-mail dialog with the

ConceptBase Development team at the University of Aachen, in Aachen, Germany and a detailed review of literature about the Andrew Toolkit.

E. ORGANIZATION OF THE STUDY

Aside from the introduction and a final conclusions chapter, this thesis consists of five major chapters. Chapter II elaborates on the REMAP model and its application. Chapter III discusses the methodology used in the development and incorporation of both the Position Object and the method that calls the hypermedia search engine. Chapter IV deals with document searching via the World-Wide Web. Chapter V presents an overview of the HyperPKM Indexing Gateway. Chapter VI illustrates the use of the HyperPKM interface.

II. REMAP PROJECT ENVIRONMENT

A. REMAP BACKGROUND

The focus of the REMAP project is the structured capture of design rationale and decisions, which are an important component of the history or "process knowledge" of software development projects. Recent research suggests that capturing the design rationale during the requirements engineering phase, early in the system development life cycle, can be very helpful and productive in ensuring the resulting system more accurately meets user requirements. This design rationale is typically lost in the course of designing and changing a system [Ref. 3]. By providing a comprehensive picture of the software development process however, the REMAP model increases the understanding of the design process and "offers a mechanism for propagating changes in the design decisions into changes in design solutions" [Ref. 4]. System designers, maintainers and users can use the process knowledge captured by the REMAP model to:

- provide design support by facilitating inter-development group communication and information exchange regarding requirements, issues, decisions, constraints, etc. [Ref. 3].
- reduce system maintenance efforts by maintaining process knowledge at the level of requirements and design rationales [Ref. 3].
- Help end users in understanding how exactly the design-deliberation process addresses their requirements and see how requested changes cause repercussions at the design level [Ref. 3].

The REMAP model incorporates the model primitives of the Issue Based Information System (IBIS). The IBIS model, was developed by Horst Rittel and is based upon the principle that the design process for complex problems is basically a conversation among the stakeholders of a project. In this process, the stakeholders use their collective expertise and viewpoints to obtain resolution of the design issues. IBIS uses a set of three design primitives and the relationships among them in a rhetorical model to represent the

"argumentation" process [Ref. 5]. This set of design primitives includes Issues, Positions and Arguments (see area enclosed by dashed line in Figure 1). An Issue is a question or concern that must be answered before problem solving can continue. A Position is a possible answer or statement that responds to a particular Issue. An Argument is a reason for supporting or objecting to a particular Position. A specific Argument may either support or object to one or more Positions [Ref. 5]. The IBIS model has been used at the Microelectronic Computer technology Center (MCC) in the Design Journal research project as a way of representing design deliberations in large design projects [Ref. 6].

The IBIS model was meant to capture the conversations and deliberations among the stakeholders in complex problems. However, it "does not recognize the context in which argumentations occur, nor the outcomes of the argumentations" [Ref. 3]. The REMAP model, therefore, incorporated additional primitives (see Figure 1) to specifically address this deficiency. These are: **Requirement, Assumption, Decision, Constraint and Design Object**. Since a system is typically designed to satisfy some end user's requirements, the Requirement primitive represents the needs/wants that the users want the system to satisfy. Additionally, since user's requirements tend to change over the life of the design project, the REMAP model includes the flexibility to modify Requirements and begin the iterative design process. Assumptions are included in the model to provide a basis for evaluating the applicability or validity of an Argument represented in the model. Decisions represent the selection of a given Position that then responds to an Issue and leads to the resolution of the given Issue. System designers then establish criteria and/or constraints that must be incorporated in the final design. The design solution is represented by the Design Object.

B. REMAP CONCEPTBASE IMPLEMENTATION

The REMAP model is implemented in *ConceptBase* -- a deductive object management system which was developed at the University of Passau [Ref. 7]. The *ConceptBase* uses the Telos knowledge representation language, which is a high level object oriented modeling language. *ConceptBase* provides a coordination mechanism that operates within a client server architecture and can be distributed over local or wide area networks using the

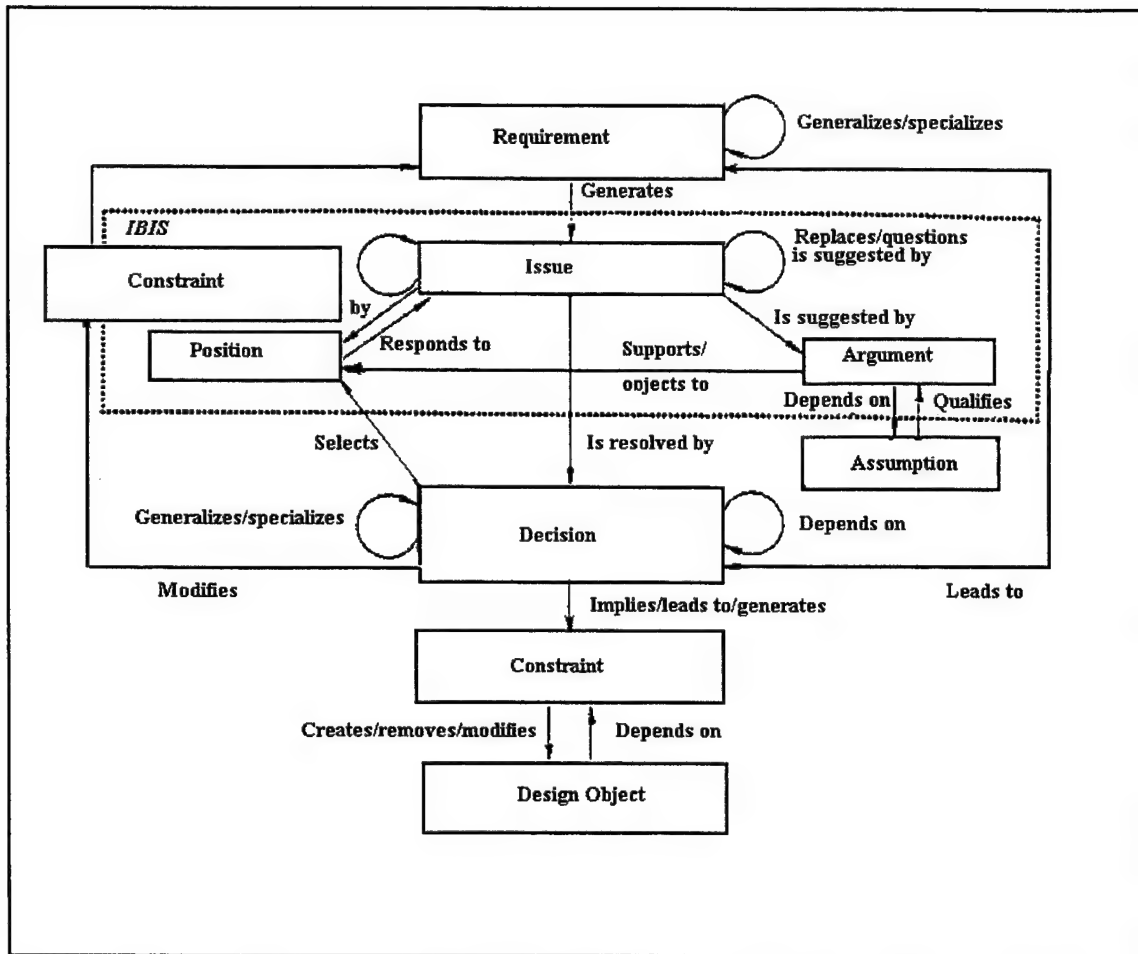


Figure 1. REMAP Conceptual Model

Internet protocol. The REMAP *GraphBrowser Utility* program implemented in the *ConceptBase* is written in the C programming language using Andrew Toolkit. This utility is designed to provide a readily customizable front end GUI for the knowledgebase of *ConceptBase*.

The *GraphBrowser Utility* (GBU) is a windows oriented interface which allows the properties and contents of any displayed object to be graphically viewed by pointing and clicking on the object with the left mouse button. These objects and their contents are displayed by the GBU as a directed acyclic graph. Various menu choices within the utility actuate queries that retrieve object instances and attributes from the knowledge base.

III. EXTENDING THE REMAP GRAPHBROWSER UTILITY (REMAP GBU) PORTION OF HYPERPKM

A. INTRODUCTION

The primary focus of this chapter is to discuss the tools and methodology that were used to create the position object extension of REMAP GBU and develop search queries for (and links to) related hypermedia design rationale documents which may exist on the WWW. The Andrew ToolKit, which was used in the implementation of REMAP GBU, various modules and class structures and hyper-link capabilities are discussed.

B. ANDREW TOOLKIT (ATK) OVERVIEW

1. Introduction

ATK was initially developed in 1982 as a joint venture between the Carnegie Mellon University and the IBM Corporation at the Information Technology Center. ATK consists of three main components:

- The Andrew Messaging System.
- The Andrew Help System.
- The ATK and application programs.

Our discussion will deal specifically with the ATK and the object oriented environment which it provides.

2. Object Oriented Programming Environment

As mentioned in Chapter I, the ATK is written in the C programming language and also provides a preprocessing environment that enables an object-oriented programming environment and dynamic linking of code. ATK greatly simplifies the creation of complex hypermedia applications through the subdivision of the program into manageable parts. Large programs are written as a large number of components, each of which describes some

behavior of some object. These objects interact to create a fluid and interactive program [Ref. 8].

This behavior is accomplished in the ATK by use of the Andrew Class System. The Andrew Class System was modeled after the C++ object-oriented environment and permits the definition of class procedures and class methods (functions) [Ref. 9]. The Andrew Class System provides a library of standard Andrew classes and a preprocessor. Additional user-defined Andrew classes are created by making two additional files: the standard C file (*.c) containing the class data and methods (functions), and a class header file (*.ch) which contains the class specification. Upon compilation of the class data and header files using a Makefile, the Andrew preprocessor generates two additional files. These files include the import header file (*.ih) which is used when any other component needs to use the class (inherit behaviors), and the export header file (*.eh) used when defining a class.

C. REMAP MODULES AND CLASS STRUCTURES

The following sections provide a brief introduction to the primary ATK and REMAP GBU application modules and classes that provide the requisite behaviors to REMAP GBU:

1. Rmain Module

The *rmain.c* file provides the standard C *main()* routine that performs the basic initializations and creates the application object. It also performs a static load of all non-ATK resident classes that are specified in the code.

2. Modul Header

The header file *modul* contains macros and defines that may be used by any class or module when it is included in that class' definition file. Additionally, it exports a pointer to the application class *appREMAP*, which allows easier access to the class methods and functions. Also, the primary macros that are included in *modul* provide for display of information and error messages.

3. Rapp Class

This is an instantiation of the ATK resident *application* (parent) class. It provides for inheritance of the *ParseArgs*, *Start*, *Run*, *Stop* and *Fork* methods for use as needed in the

application. The *rapp* class header, *rapp.ch*, holds pointers to the parameters that start the program when called by *rmain*. They are initialized by the method *ParseArgs* which is called in the *rmain* module. This class also initializes the ConceptBase and the graph data object when *rapp_Start()* is called.

4. RCB Class

The *rCB* class is the primary ConceptBase interface, and handles all of the communications with the ConceptBase server. It includes methods that allow connects and disconnects with the server and provide communications via ask and tell statements.

5. Rgraphv Graphview Class

The *rgraphv* class is an instantiation of the ATK resident class *graphview*. It is a specialization of the *graphview* class since it over-rides the inherited methods, such as *graphview_Hit()*, which performs specific actions when an object is clicked on by the mouse. Additionally, it provides its own menus and a means for updating and maintaining its menus.

Menu items within REMAP GBU are not implemented as class methods, and are more similar to functions which could be located anywhere. Within this application, the functions are placed in class definition files to which they are logically related. For example, when developing the position object's *issueobjCom_CreatePositionForIssue()* it is placed in the *issue* class file since it is logically related to issue.

6. Robject Class

The *robject* class file is a generic super class definition which provides basic class behaviors and methods as well as inheritance for all objects which are displayed in the graphview. In version 1.7 there are three object classes implemented: Requirements, Issues and Positions. These object instantiations require the super classes methods to, at a minimum, create, display and initialize each of the object instantiations.

7. Requirement Class

This class is a subclass of the *robject* class. It includes inherited methods which instantiate the requirement object, definitions which specify its shape and appearance within the browser and other methods which define its interactions with the issue class.

8. Issue Class

This class is a subclass of the *object* class. It includes inherited methods which instantiate the issue object, definitions which delineate its shape and appearance within the browser and other methods which define its interactions with the position class.

9. Position Class

This class is a subclass of the *object* class. It includes inherited methods which instantiate the position object, definitions which describe its shape and appearance within the browser and other methods which define its interactions with the argument (not implemented yet) and issue classes.

10. Sellist Selectionlist Class

The *sellist* class provides a window for selecting a single choice from a list of displayed elements. It inherits its behavior from the ATK resident class *suite*. Selectionlist uses a linked list of pointers which point to strings (char *) that you wish to display.

D. REVISION CONTROL

1. Purpose of Revision Control

Software revision control provides a means of managing the configuration or change process of a software application over its life cycle. It is especially useful case of developing REMAP GBU since it provides a means of recording the history, progress and milestones of the application's development. It also permits a method by which older versions may be restored to regain a certain functionality or perform a demonstration of a previous versions' capabilities.

2. The Revision Control System (RCS)

The GNU RCS is available on the network and "isr1" workstation where all the primary work for REMAP GBU was done. Commands which were most frequently used while developing this prototype include co(l), ci(l), rcs(l) and rcsdiff(l). The "man" pages or any number of UNIX programming references contain indepth descriptions on the use of these commands.

3. Managing REMAP GBU Revisions Using RCS

The development project was begun by creating the new version directory "v1_7" and then checking out the code that was last logged in to the RCS database. The sequence of commands to do this are as follows:

<i>cd remap</i>	(change directory to where code exists)
<i>mkdir v1_7</i>	(creates the new prototype directory)
<i>ln -s ../src/RCS</i>	(creates a soft link to RCS database path)
<i>co -r RCS/*.c* RCS/*.h* RCS/Makefile</i>	(checks out last version and releases the lock for all code files needed)

To check in files to the RCS database while still working on the current version, the following are used:

<i>ci -rfk1.7 *.c *.ch *.h</i>	(checks in *.c, *.ch and *.h files without locking, forcing the checkin even without changes to file and searches checking file for information keywords containing author, change time, state, etc., to place in the RCS database)
--------------------------------	---

After entering this command, a short login script that describes the version is entered.

When version 1.7 was completed, and no further modifications were to be made, the following command was used to check in the final version to the RCS database:

<i>ci -f1.7 *.c *.ch *.h</i>	(checks in *.c, *.ch and *.h files with lock and removes the files from the working directory)
------------------------------	--

E. CODING HYPER PKM (REMAP GBU EXTENSION)

1. Creating the Position Class

The following steps were taken to create the *position* class:

a. *Position.c*

This code for the position object included the basic methods required to initialize, destroy, create, display and return the group name of the position object. This was

created by using the same methodology used in version 1.5 (v1.5) code of issue.c. The changes needed to turn the version 1.5 issue.c code into the version 1.7 position.c code included the following:

- define the group name, which must be exactly as it will be defined in rCB.ch: *szGroupName = szPOSITION;*
- change the shape of the node to: *#define szPosObjDefaultShape "rectnode";*
- change all instances of the words Issue or Iss to Position or Pos.

b. *Position.ch*

This class header file was created by re-using the methodology used for v1.5 issue.ch. The only changes required were to change all instances of the word Issue to Position.

c. *Issue.c*

This file was modified to include the behaviors needed to make the position object fully functional. The methods GetAllPositions(), SelectOnePosition(), LoadPositionForIssue(), LoadAllPositionsForIssue() and CreatePositionForIssue were placed in the issue object since they were logically related to this object. The following were necessary to complete the modification of issue.c:

- Since the basic implementation methods would need to be accessed the position header file would have to be imported. The following command is placed in the import section of the source code:

#include "position.ih"

- In the implementation section of the source code a character pointer variable must be assigned that determines the name of the edge between the issue and position objects. This name must be exactly the same as will be defined in rCB.ch:

static char szPositionCatName =
szPOSITION_respondsto_ISSUE*

- GetAllPositions() and SelectOnePosition() were created by using a linked list structure. These two methods are needed to provide a selection mechanism to choose between a number of positions that may have been previously created and stored in the ConceptBase during a persistent session. GetAllPositions creates the linked list for a selected issue. The linked list of positions is called by SelectOnePosition which then creates a "selectionlist" which may be chosen from when rgraphv_MenusOn is called. It is extremely important to remember when working with linked list structures that you must ensure the last element of the list is assigned to "NULL".
- LoadPositionForIssue is used when an issue is selected which has a number of positions that may be displayed. This method takes the position chosen and displays along with its associated edge in the graphview. If there are no positions from which to choose, a message is displayed stating "No Position".
- LoadAllPositionsForIssue is used when the user selects this choice from the GBU menu. The method calls GetAllPositions to create the linked list of stored positions, and once the list is created traverses down the linked list to display all positions and edges.
- CreatePositionForIssue methodology is the same as that used for CreateIssueForRequirement. One significant difference is that the order of the arguments for rgraphv_InsertEdge must logically match that of the node-edge arrangement that is created. In this case since POSITION_respondsto_ISSUE is being developed, the direction of the edge's arrowhead should point from the position(source) to the issue(target). The appropriate command is:

```
rgraphv_InsertEdge(rapp_RetGraphv((CRapp)appREMAP), szCurItem,  
szLabelSelIss, issueobj_RetPositionsIssueCatName(),  
issueobj_RetPositionsIssueCatGroupName());
```

d. Issue.ch

The only modifications required for this source file are the addition of the new method and menu function prototypes:

```
RetPositionsIssueCatName() returns char*;  
RetPositionsIssueCatGroupName() returns char*;  
GetAllPositions(char* szIssueID) returns CLList;  
SelectOnePosition(char *szIssueID) returns char*;
```

```

overrides:
RetGroupName() returns char*;
};

/** these procedures will be called by menu commands */

extern void issueobjCOM_LoadPositionForIssue();
extern void issueobjCOM_LoadAllPositionsForIssue();
extern void issueobjCOM_CreatePositionForIssue();

```

e. Rmain.c

Two additions were required for this source file. They include adding *#include position.ih* to the import section and adding *positionobj_StaticLoad()* under the *rStaticLoad()* method.

f. RCB.ch

Two defines had to be created for the ConceptBase. Recall from previous discussion in *position.c* that the new position object functionality (object and objectcategory names) had to be exactly as defined for the ConceptBase. This is needed since *rCB* provides interface definitions for the ConceptBase that are also exported to be used by other source files. In this case

```

#define szPOSITION          "POSITION"

```

and

```

#define szPOSITION_respondsto_ISSUE "respondsto"

```

were needed to define the functionality that creates the position object and related edge on the graphview .

g. Rgraphv.c

The following modifications were made to *rgraphv.c* to provide to menu items required for the position object and the "Search Related Documents" function call:

- The following code was added after "remap-create-issue" to provide menu items to call the methods required to load and create positions:

```

/* _____ */
RG_MENU_NOTHING,
/* _____ */
{ "remap-load-position",
  NULL,
  NULL,
  ",Load One Position",
  NULL,
  RG_MENU_MASK_ONE_ISSUE,
  issueobjCOM_LoadPositionForIssue,
  "Load a position for the selected issue",
  NULL},
/* _____ */
{ "remap-load-all-positions",
  NULL,
  NULL,
  ",Load All Positions",
  NULL,
  RG_MENU_MASK_ONE_ISSUE,
  issueobjCOM_LoadAllPositionsForIssue,
  "Load all positions for the selected issue",
  NULL},
/* _____ */
{ "remap-create-position",
  NULL,
  NULL,
  ",Create Position",
  NULL,
  RG_MENU_MASK_ONE_ISSUE,
  issueobjCOM_CreatePositionForIssue,
  "Create a new position for selected issue",
  NULL},

```

- The following code was inserted after the "Show Contents" menu selection to create to make the "Search Similar Documents" menu selection:

```

/* _____ */
RG_MENU_NOTHING,
/* _____ */
{ "remap-search-similar-documents",
  NULL,
  NULL,
  ",Search Similar Documents",
  NULL,
  RG_MENU_MASK_ONE_OBJECT,
  robjectCOM_ShowSimilarDocuments,
  "Search WWW for similar contents of a selected object",
  NULL},
/* _____ */

```

```

RG_MENU_NOTHING,
RG_MENU_NOTHING,
/* _____ */

```

Adding code to create the additional menu functionality was simply a task of reusing segments of the current code. Once a similar segment was identified, the code was copied to the new menu list location and modified to suit the required task.

- In the RetSelectState() method lines of code had to be added to account for various position object counters and variables. On all lines where *iReq* and *iIss* were initialized "iPos" had to be added. Additionally, a line had to be added to account for incrementing a counter when the RetClassName() method did a string comparison for position objects.

```

if (strcmp(szGroup, positionobj_RetClassName()) == 0)
    iPos++;

```

A "Menu_Mask" also had to be provided to account for position objects.

```

/* only one position selected */
if (iReq==0 && iIss==0 && iPos==1)
iNewMask |= RG_MENU_MASK_ONE_OBJECT | RG_MENU_MASK_ONE_
POSITION;

```

h. Rgraphv.ch

The only modification to rgraphv.ch needed was the addition of a #define line to define the state of the menu "Mask_One_Position" item.

```

#define RG_MENU_MASK_ONE_POSITION    (1<<4)

```

2. Creating the Search Related Documents Class Method

This method is implemented as class method within rgraphv.c. It uses a method ShowSimilarDocuments(), which when activated by a mouse hit on the menu selection "Search Related Documents", will fork a new process and display the hypermedia search form. The method "Search Related Documents" is defined in robject.c source code. To perform the coding of this method, the following were required:

- In the implementation section define a pointer to a character variable with the proper Uniform Resource Locator (URL): *static char* szObjectContentsDisplayURL = "http://sm.nps.navy.mil/cgi-bin/ice-form.pl";*
- Change the browser to Netscape by changing the ObjectContentsDisplayTool variable to: *static char* szObjectContentsDisplayTool = "netscape"*
- Create the method by using the basic methodology of the rObjectCOM_ShowContents method. Required actions of the code were to fork a new process (create a window with the Netscape browser application launched in it) and call the hypermedia search form's URL after the application was launched. The code required to perform this is:

```

void robjectCOM_ShowSimilarDocuments(CRGraphv gv)
{
int pid;

/* fork a new process */
if ((pid = fork()) > 0) {
    message_DisplayString(rapp_RetGraphv
        ((Crapp)appREMAP), 10, "Similar Documents
        Search Form will be displayed...");
    return;
}
else {
    /* run called program */
    execlp(szObjectContentsDisplayTool,
        szObjectContentsDisplayTool,
        szObjectContentsDisplayURL, (char *)0);
}
}

```

- Robject.ch needed to be modified to include a menu command procedure prototype line to define the function "Show Similar Documents."

```
extern void robjectCOM_ShowSimilarDocuments();
```

F. COMPILATION OF HYPERPKM

1. Setting Environment Variables

The environment variables establish the paths which are necessary to complete the compilation of the source files and dynamic objects as well as establish the required

dependencies. These variables are set in the .cshrc file of the project directory and should be set as shown below:

```
setenv ANDREWDIR /usr/local/andrew51  
setenv PATH .:$ANDREWDIR/bin:$PATH  
setenv CLASSPATH .:$ANDREWDIR
```

2. Creating the Makefile

The Makefile for this prototype was created by adding the required position object dependencies. These are the same dependencies as required for all REMAP model primitives, and help to define the basic parameters and objects which are necessary to define the objects. A few of the required statements include:

```
position.o: position.eh ../include/class.h /usr/include/stdio.h robject.ih  
position.o: ../include/shape.ih ../include/etc.h ../include/types-atk.h  
position.o: /usr/local/andrew51/include/atk/rect.h  
position.o: /usr/local/andrew51/include/atk/point.h modul.h  
position.o: ../include/attrib.ih ../include/list.ih  
position.o: /usr/local/andrew51/include/atk/fontdesc.ih  
position.o: /usr/local/andrew51/include/atk/graphic.ih  
position.o: /usr/local/andrew51/include/atk/observe.ih  
position.o: /usr/local/andrew51/include/atk/pixelimg.ih ../include/gitem.ih  
position.o: ../include/attrib.h sellist.ih  
position.o: /usr/local/andrew51/include/atk/im.ih  
position.o: /usr/local/andrew51/include/atk/view.ih
```

This process was simplified by using xedit or emacs to copy the issue dependencies and then substituting position for issue.

3. Makefile Generation and Code Compilation

When first compiling the code, making changes by adding dependencies (# include statements) to the source files or any time after changes have been made in *.h or *.ch files, you must use the command:

```
make headers depend rg
```

For subsequent compilations, you just need to enter:

```
make
```

IV. SEARCHING VIA THE WORLD-WIDE WEB

A. EXISTING SEARCH OPTIONS

Recently, the use of the World-Wide Web has caught the attention of the DOD, so much that, many organizations throughout DOD now have their own Home Pages on the Web. This new method of providing and locating information has led to enormous amounts of traffic being generated on military networks. Much of this traffic is generated by users trying to located relevant information that is required for day to day operations. The DOD, has in the past, relied heavily on Gopher (http://sm.nps.navy.mil/webmaster/guide/eeg_186.html#SEC187) and FTP (http://sm.nps.navy.mil/webmaster/guide/eeg_138.html) sites to store and retrieve data from its various organizations. Users, would use search engines such as Archie (http://sm.nps.navy.mil/webmaster/guide/eeg_139.html) and Veronica (http://sm.nps.navy.mil/webmaster/guide/eeg_187.html) to search these sites for relevant information. In the way the Web is currently being used, these search engines are obsolete.

“Existing searching techniques on [the] WWW fall into two main categories: hypertext browsing and keyword searching (and a combination of the two)” [Ref. 10]. Browsing the Web involves “linking” or “surfing” from site to site. If the user is searching for information in this fashion, ...“then the user must know the meaning of very broad terms, and be able to judge where the specific information of interest falls under those terms.” [Ref. 10] When utilizing hypertext browsing in order to locate information, users can become lost in hyperspace or experience “information overload” due the size of the Web.

Browsing is the common interaction paradigm for hypertext, when a user is gathering information. It is very useful for reading and comprehending the contents of a hypertext, but not suitable for locating a specific piece of information [Ref. 11].

Moreover, keyword searching allows the user to search for a word, partial words, combination of words, phrases, or words with boolean terms such as AND, OR, or NOT.

“Keyword searches typically make use of a pre-compiled index (inverted list) which contains an entry for each word that has pointers to all documents containing that word” [Ref. 10].

B. INFORMATION INDEXING AND SEARCHING

Although the World-Wide Web provides a remarkably opulent foundation of information, it does not support a consistent and efficient means of retrieving specific information based on user-defined queries [Ref. 11]. Insofar, many types of search engines have been developed for the World-Wide Web, such as, server-side indexers and “Robots” or “Spiders”. A pre-compiled index or database is utilized by the search engine in order to answer the users queries. This index can either be site-specific or Web-specific. A Web-specific index is generated through the use of “Robots” or “Spiders”. The “Robot” automatically travels the Web in search of new sites or information. When a new site or information is found, a site index is generated and returned to the originating host of the “Robot”. This index is then compared to the master Web index on the host to see if this site already exists. If the site exists, then the master Web index is updated with whatever new information the site might contain. If not, then the site is added to the master Web index. The problem with this approach is that it takes a very long time for the “Robot” to travel the Web in search of new information. Once found and indexed, a site might not be revisited by the “Robot” in an acceptable amount of time. This leads to “dated” and sometimes useless information.

The nature of the WWW presents an unusual problem for building indexes. Since there is no control over when and how documents are added to the systems, there is no way to ensure that they are added to an index. This problem is further complicated when the document is modified. This problem is addressed somewhat by the use of robots and spiders such as Lycos [(http://lycos.cs.cmu.edu/)] and WebCrawler [(http://webcrawler.com/)]...But such programs place a heavy burden on network resources, particularly since they must search the network repeatedly to find updated materials (both new and revised) [Ref. 10].

Furthermore, many sites sit behind firewalls or are unknown to the rest of the Web, therefore, a "Robot" will not locate them.

On the other hand, the concept of site-specific indexes is one in which an index is generated by the site host. "By supplying a pre-computed index of keywords, a fully indexed server eliminates the need for automatic indexers (such as web robots or spiders) to walk the entire server tree, which is an unnecessary waste of resources" [Ref. 12]. The index is accessed via a gateway or interface over the World-Wide Web in which users can submit queries to the index or database.

Some World-Wide Web servers already implement keyword searches via an interface to [indexes such as] WAISINDEX [(http://sm.nps.navy.mil/webmaster/guide/eeg_212.html)]. However, this approach lacks many important features that free text search engines provide, and does not support remapping of physical directory structures to virtual paths [Ref. 12].

The remapping of physical directory structures to virtual paths is necessary for hypertext to work via a World-Wide Web gateway, thus allowing the user to access, via hypertext links, whatever information was indexed on a system. The index can be updated automatically anytime new information is added to the hosts' system. This is the approach that is taken by the HyperPKM model.

C. IMPROVEMENTS TO KEYWORD SEARCH ENGINES

Built into the HyperPKM model is the ability to perform keyword searches by incorporating and supporting substring matching, proximity searching and thesauri based queries. Individually, these features are not new to keyword searching, but when all are used in conjunction with a site-specific index and a specific technical thesaurus, a user would be able to locate all information the site contained that was relevant to the user's queries.

V. OVERVIEW OF THE HYPERPKM INDEXING GATEWAY

The use of the HyperPKM model requires the use of site indexing based on certain criteria. In order for an index to work efficiently and provide relevant data to the user, an index of the documents contained on the system must be generated. This index is a categorized listing of all the relevant words located within the documents. Limiting the size and relevance of the index and being able to provide useful information to the user is an item of concern that is addressed.

The HyperPKM model also incorporates a specific technical thesaurus that allows the user to access a database of terms that are related to the queried keyword(s). This feature allows for the broadening and narrowing of related search terms, thus, providing the user with an increased ability to locate desired documents.

Accessing the HyperPKM model is done through the use of a graphical interface between the user and the server via the World-Wide Web. This provides for the ability of any computer system utilizing any Web browser to access the documents contained on the host system.

A. SITE INDEXING

Various considerations must be dealt with in creating an index of a system and being able to perform searches on the index. A database or archive may contain a myriad of documents that when indexed would create an index of substantial size. Thus, system resources and the time required to perform a search become factors.

The considerations of site indexing and searching that must be undertaken are: (1) stop wording, (2) word frequency, (3) abbreviations and acronyms, (4) word stemming, and (5) boolean searches.

1. Stop Wording

One purpose of the HyperPKM indexing model is to allow users to differentiate between the differing documents returned as a result of a query. Various documents that contain many of the same or "common" words are not useful for differentiation. This

problem can be solved through the use of Stop Wording. Stop Wording can be accomplished automatically by the indexer in two different ways. The indexer can define a word as “common”, that is, the word is contained in over a large percentage of all indexed files, or the indexer can access a Stop Wording list. A Stop Wording list is constructed manually and can be updated as the need arises. The list would contain words that the system administrator felt were too common to be of use for search and retrieval. These words would not be indexed, nor would they show up as a result of a query for such words.

2. Word Frequency

Word frequency is the number of occurrences that a word is located in a document. Through the use of an inverted index, each documents’ word count is tabulated and stored in the index. This feature gives the user some impression as to the relevance of the keyword to the document itself.

3. Abbreviations and Acronyms

Throughout the DOD, terms that are abbreviations and acronyms of words are very prevalent. These terms have to be dealt with in a predefined way. If a query is conducted for “IT”, an abbreviation for “Information Technology”, then the indexing system must know the difference between “IT” and “it”. Otherwise, the user would be presented with an overwhelming number of documents that contain the term “it” (unless “it” is contained in the common or stop wording list). The HyperPKM model assumes that any abbreviation or acronym would be represented by all capital letters. If a term with two or more capital letters is encountered while indexing, the term is considered to be all capitals. For an abbreviated term or acronym to be located during a query, the user must enter the term in all capital letters.

4. Word Stemming

Word stemming (also known as substring matching or fault tolerant retrieval) allows for the search and retrieval term to be broadened. This feature allows the user to input a keyword and for the server to return all variances of the word (i.e., retrieve = retrieves = retrieved = retrieving) The HyperPKM model accomplishes this through the use of the Levenshtein algorithm [Ref. 13].

5. Boolean Searches

Boolean searches allow for the query to consist of more than one keyword. Through the use of boolean operators (i.e., and, or, not, near), the user can query the index to locate documents that contain certain combinations of words. Such operators would be utilized as follows:

- Information and technology (both terms must be in document).
- Information or technology (either term must be in document).
- Information not technology (only "information" in document).
- Information near# technology ("information" must be within # words of "technology" -- also known as proximity searching).

The HyperPKM model utilizes these boolean search operators and can be setup to assume either the "and" or "or" operator in the event two or more keywords are entered without the use of any boolean operators.

B. THESAURUS

A thesaurus is considerably more than a list of synonyms. "It is a semantic network containing concepts that are related to one another in various ways" [Ref. 12]. The HyperPKM model employs a specific technical thesaurus that utilizes the ANSI (American National Standards Institute -- <http://www.ansi.org/>) standard **Thesaurus Image Format (TIF)**. In this manner, a query is not only a search for a keyword, but a conceptual search for word meaning and topic area. To utilize this feature, a specific technical thesaurus that pertains to the documents/keywords at hand must be created.

One way to develop a specific technical thesaurus is to generate one automatically through the use of computer programs and scripts [Refs. 14, 15, 16, 17, and 18]. However, a thesaurus that is generated automatically by computers produce low precision levels in respect to ones developed by people. A thesaurus developed by humans realize a 77-98% concept precision level, whereas, the one's generated automatically by computers realize a

24-37% precision level [Refs. 14 and 15]. In the case of the HyperPKM model, a specific technical thesaurus was created manually from all available sources of information.

C. WORLD-WIDE WEB INTERFACE

The HyperPKM Indexing Gateway is a CGI (Common Gateway Interface -- <http://hoohoo.ncsa.uiuc.edu/cgi/intro.html>) compliant interface. Through the use of host server software, such as the HTTPd public domain server software from the National Center for Super Computing Applications (NCSA -- <http://hoohoo.ncsa.uiuc.edu/docs/Overview.html>), a gateway can be created to provide documents and files to Web browsers such as the Netscape Navigator™ (http://www.mcom.com/comprod/netscape_nav.html) or Mosaic™ (<http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html>). This means that through the use of the HyperPKM Indexing Gateway, the responses to users queries are dynamically generated, created on the fly and executed in real time. The user inputs what is required of the system and the system responds and generates a reply. A CGI program such as the HyperPKM Indexing Gateway is in essence, a program that is open for all to use.

The CGI program provides for the utilization and access to the host HTTPd server via the World-Wide Web. Scripts written in a language called PERL (<http://www1.cis.ufl.edu/perl/>) are used to process the users inputs. These scripts are run as a background operation to the actual HyperPKM Gateway Interface. The user inputs a query and then the PERL scripts are invoked. The user's queries are broken down into subsections based on what the user selected on the gateway interface form. Keywords that are entered are compared to the site-specific index. Search operators, if any, are identified and used. Finally, if selected, the specific technical thesaurus is interpreted for any related words to the queried keywords. Then the PERL scripts generate a HTML document listing all documents that conform to the users queries. This generated document does not exist on the host system, but is created only for the current user. It contains the following information based on the users inputs:

- The keyword(s) that the user entered.
- The context or document hierarchy that was searched.
- Any search operators that were selected or entered by the user.
- Utilization, if selected, of the thesaurus and/or substring matching.

The document also contains items, based on what was discovered in the site-specific index, that are relevant to the users query, such as:

- Titles of documents that are related to the search query.
- File date or last change date (for determination of currentness).
- The directory or subtree of hierarchy in which the document is located.
- Actual filename.
- The keyword(s) or related words that were found.
- The number of occurrences of each word found.

Since the documents are dynamically generated, virtual paths are created in order for hypertext links to be utilized. The title of the document becomes the hypertext link so that the user only has to select the desired document and then is "linked" to the document of interest. The document is then displayed for the user to peruse.

VI. EXAMPLE OF THE HYPERPKM INDEXING GATEWAY

In this chapter, the use of HyperPKM is illustrated as a scenario in which a REMAP object is linked to the contents of a large volume of documents governing the procurement of government systems. These documents are part of a DOD acquisition manual which is used by the military in fulfilling the mission needs in terms of acquisition.

The DOD acquisition manuals consist of numerous documents that are related to the time-consuming process of the procurement of mission essential components. In using these manuals, the understanding of the interrelationship between documents is essential to project completion and knowledge of numerous key terms and acronyms is imperative. Being able to locate vital information is a necessity, yet, is often a very time intensive and futile process. The need for a search and retrieval engine that can be tailored to the specific requirements of the acquisition process is dealt with by the HyperPKM model. During the development of systems for the DOD, it is important for a designer to access information relevant to the project that may be contained in acquisition documents. DOD acquisition manuals consist of numerous volumes of information. Many project tasks are conducted offsite, away from the project office. Laptop computers help to serve as an intermediary way to alleviate the problem of transportability of the acquisition documents. However, they have limitations. Usually, due to capacity and search constraints, only the required documents dealing with the task at hand are loaded onto the laptop computers. Many times, the need for additional documents is required. However, due to time constraints and locality, the notion of returning to the home office is not an option. Thus, the avenue for the utilization of the World-Wide Web. Through the use of the laptop computer, a phone line or a cellular phone, the project

member can contact the home office server via a dial-up account, execute an X-Window session and by starting the REMAP GBU portion of the HyperPKM model, continue work on an ongoing design project as well as search for and retrieve any pertinent information. To begin this demonstration, let us assume that a design project includes user *Requirements*, a generated *Issue* and a *Position*. The Requirement of interest specifies that the design project must be within budget constraints. The related Issue concerns which budgeting and planning method should be used. Our Position relates that our organization must use the Planning, Programming and Budgeting System (PPBS) method. To start the on-line session, the user starts the REMAP GBU and creates the Requirement, Issue and Position as shown in Figure 2. To search for additional supporting information for the Position that may exist on the WWW, the user selects the position Use_PPBS with the mouse and then selects the Search Related Documents option (shown on Figure 2). This selection activates the Netscape™ browser and accesses the HyperPKM Indexing Gateway.

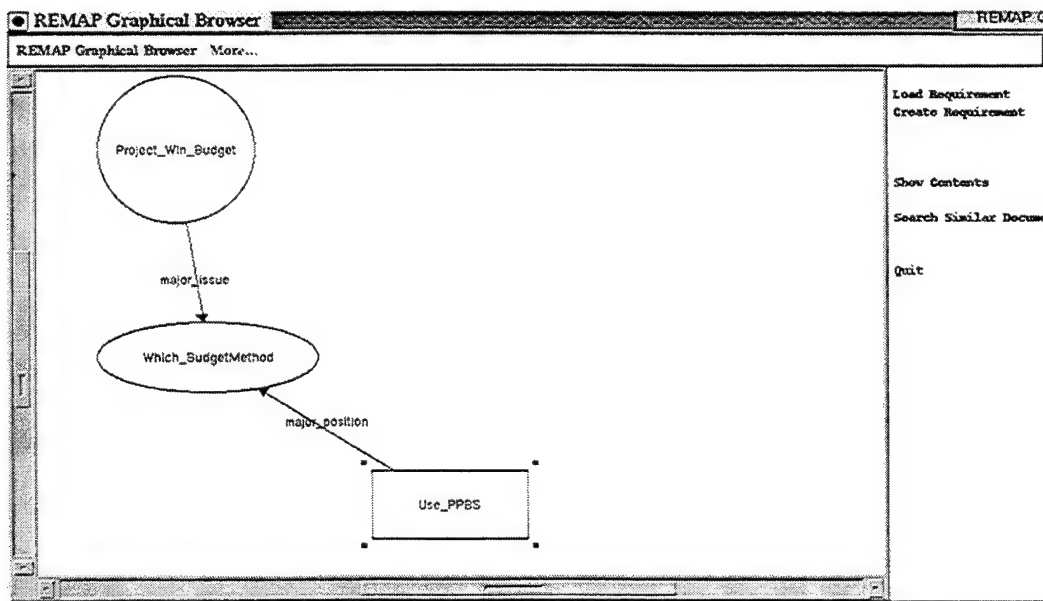


Figure 2. REMAP GBU Session

The HyperPKM Indexing Gateway is a user friendly interface between a client and server via the World-Wide Web. It allows for search and retrieval with the minimal of effort on the part of the user. An illustrated example is portrayed showing the search and retrieval

capabilities of the HyperPKM model and a DOD acquisition manual as the source of indexed information. Figure 3 depicts the HyperPKM Indexing Gateway as seen with the Netscape Navigator™. The gateway is built upon a modified version of the ICE search engine designed by Christian Neuss of the Fraunhofer Institute for Computer Graphics (<http://www.igd.fhg.de:80/>).

Data entry fields are made available to the user to enter the following data:

- Keyword(s).
- Boolean search operators (and, or, etc.).
- Date relevancy searches (check for revised documents).
- Thesauri based searches.
- Substring searches (word stemming).
- Document hierarchy.

Document hierarchy is an important feature in that it allows the user to limit the search criteria to a predefined subset of the total document structure. This feature helps to narrow down the search area and prevents information overload by having too much information presented to the user.

The acquisition process hierarchy is broken down into a total of six phases. As shown in Figure 4, the user can select to search either the entire acquisition manual document or any one of the individual phases.

Netscape - [HyperPKM Indexing Gateway]

File Edit View Go Bookmarks Options Directory Help

Back Forward Home Reload Images Open Print Find Stop

HyperPKM Indexing Gateway

Type in the keyword or several keywords connected with "and" and "or".
Example: "picture and binary".

You can restrict searches to documents that have changed during the last n days
(leave the other fields empty to get all documents changed during that time):
Number of days:

Turn on use of thesaurus to extend a search to all synonyms of a term, turn on
substring matching to extend searches to words which contain the given term as a
substring:

☐ Use thesaurus ☐ Substring matching

Limits search to a subtree of the document hierarchy:

Search all acquisition documents

Start search:

Reset to default values:

Figure 3. HyperPKM Indexing Gateway

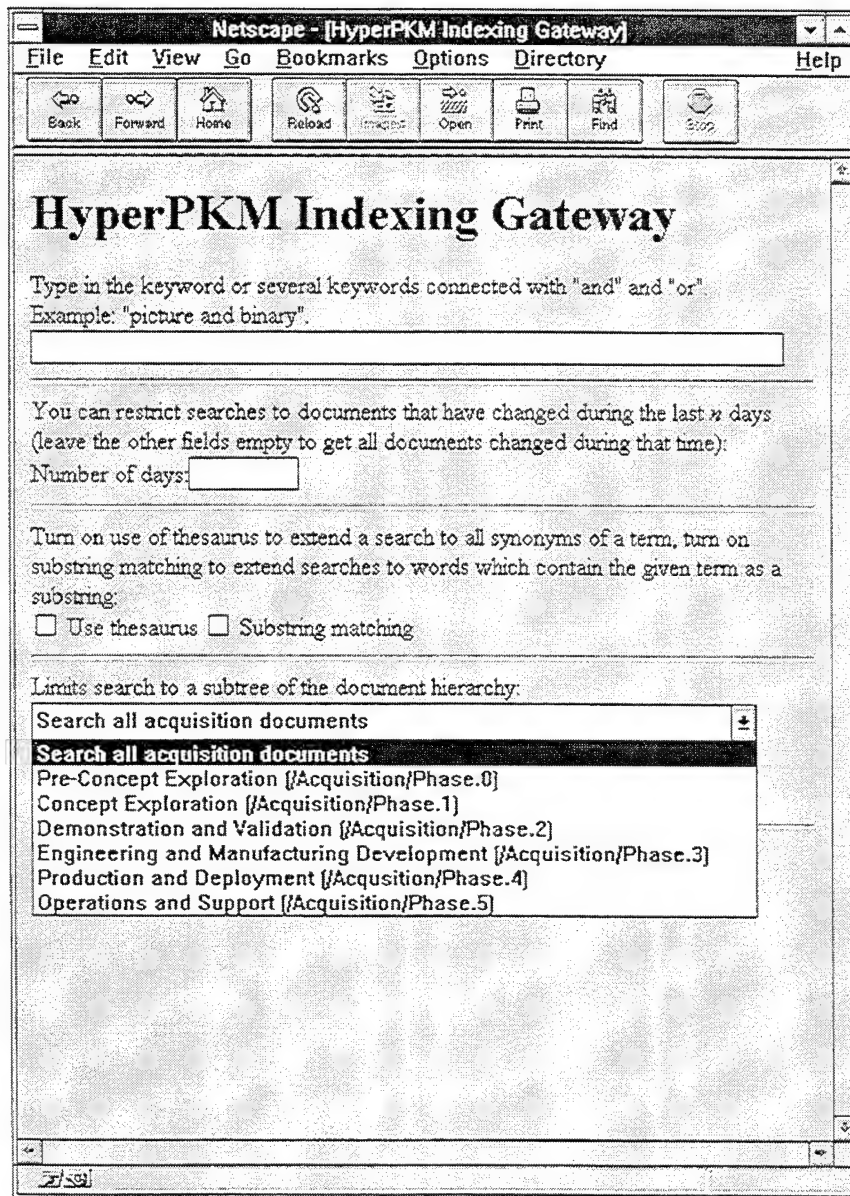


Figure 4. Searchable Document Hierarchy

To illustrate the example, the search term of "PPBS" (Planning, Programming and Budgeting System) has been entered into the keyword search field (see Figure 5). The document hierarchy field has been set to "Pre-Concept Exploration" or Phase 0 on the notion that the user is searching for information dealing solely with this area of the acquisition process (see Figure 5). This action will greatly reduce the amount of superfluous information that is generated and will keep the data manageable for the user.

The screenshot shows a Netscape browser window titled "Netscape - [HyperPKM Indexing Gateway]". The menu bar includes File, Edit, View, Go, Bookmarks, Options, Directory, and Help. Below the menu bar is a toolbar with icons for Back, Forward, Home, Reload, Images, Open, Print, Find, and Stop. The main content area displays the "HyperPKM Indexing Gateway" title. Below the title, there is a text input field for keywords with the example "picture and binary" and the entered term "PPBS". A section titled "You can restrict searches to documents that have changed during the last n days" includes a "Number of days" input field. Another section titled "Turn on use of thesaurus to extend a search to all synonyms of a term, turn on substring matching to extend searches to words which contain the given term as a substring" includes checkboxes for "Use thesaurus" and "Substring matching". A section titled "Limits search to a subtree of the document hierarchy" includes a dropdown menu showing "Pre-Concept Exploration [Acquisition/Phase.0]". At the bottom, there are "Start search: Ok" and "Reset to default values: Reset" buttons.

Figure 5. Keyword and Selected Hierarchy Data Field

Upon the selection of the "Start Search" button by the user, the HyperPKM Gateway Interface calls the associated PERL scripts to perform the search of the site-specific index. Any related information that is contained in the index is retrieved and parsed, and then it is transformed into a dynamically generated HTML document complete with virtual remapping of the documents' location (see Figure 6). A search conducted in this manner for the keyword "PPBS" under the Pre-Concept and Exploration phase resulted in only two documents being found (see Figure 6).

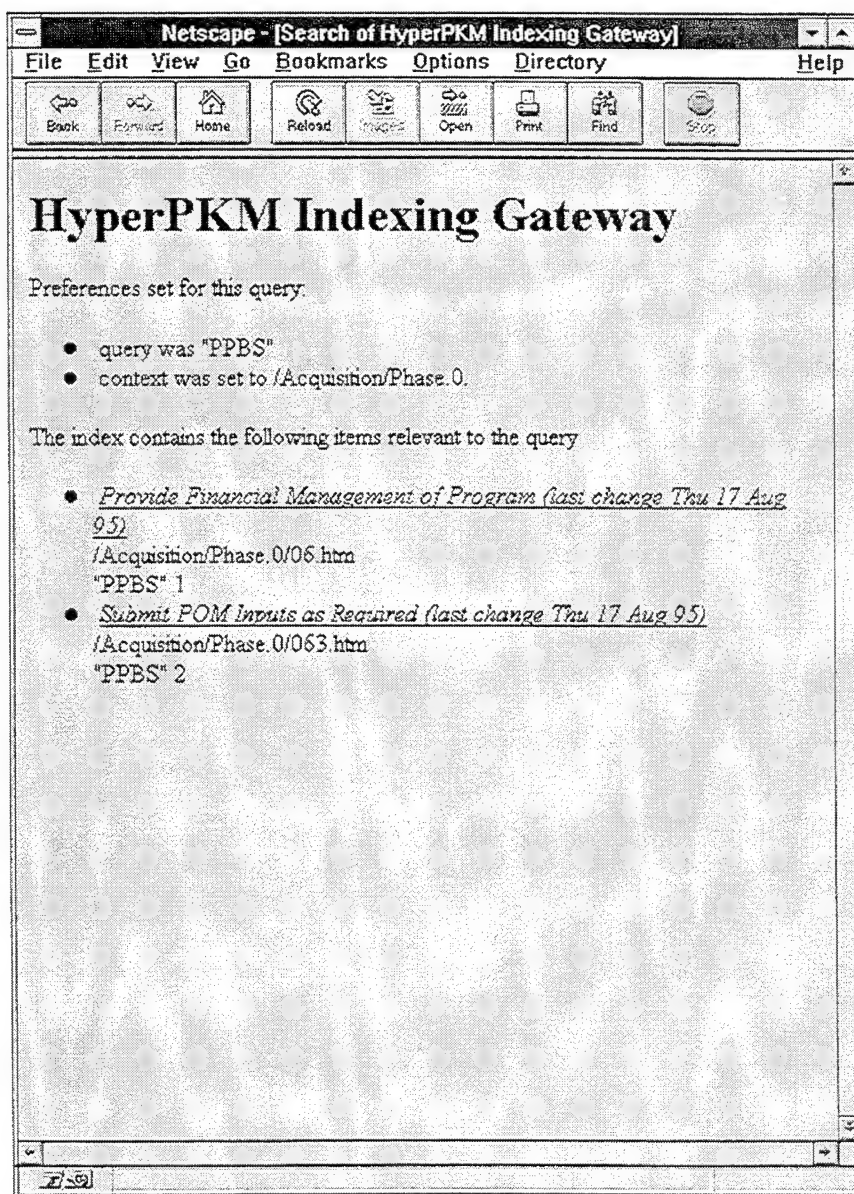


Figure 6. Document Resulting From Search Query

The user could also enter the keywords "planning and programming and budgeting" (see Figure 7) in which case the search would return a total of three documents that are related to the query (see Figure 8).

The screenshot shows a Netscape browser window titled "Netscape [HyperPKM Indexing Gateway]". The menu bar includes "File", "Edit", "View", "Go", "Bookmarks", "Options", "Directory", and "Help". Below the menu bar is a toolbar with icons for "Back", "Forward", "Home", "Reload", "Images", "Open", "Print", "Find", and "Stop". The main content area displays the "HyperPKM Indexing Gateway" title. Below the title, there is a text input field containing the search query "planning and programming and budgeting". Above this field, instructions state: "Type in the keyword or several keywords connected with 'and' and 'or'. Example: 'picture and binary'." Below the search field, there is a section for restricting searches by the number of days, with a text input field for "Number of days:". Further down, there are two checkboxes: "Use thesaurus" and "Substring matching", both of which are currently unchecked. Below these checkboxes, there is a section for limiting the search to a subtree of the document hierarchy, with a dropdown menu showing "Pre-Concept Exploration [/Acquisition/Phase.0]". At the bottom of the form, there are two buttons: "Start search: Ok" and "Reset to default values: Reset".

Figure 7. Alternative Search Technique

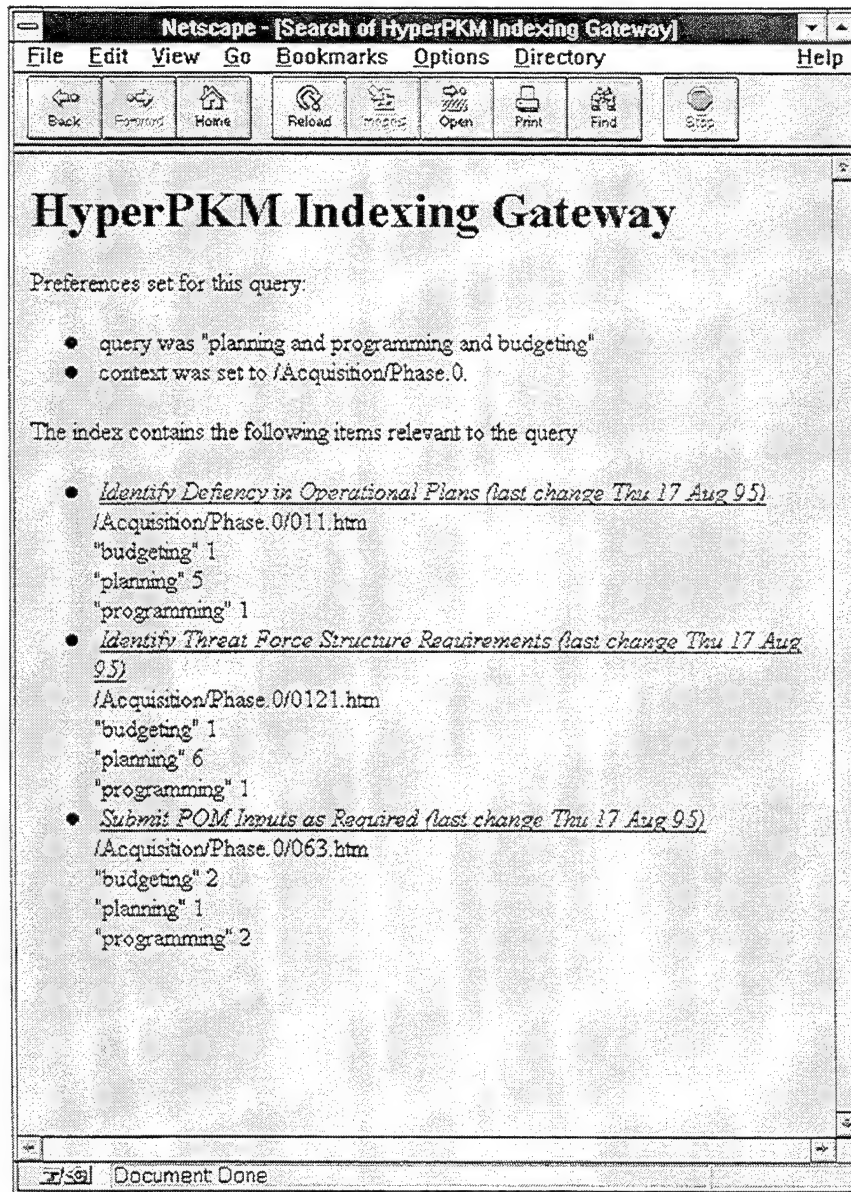


Figure 8. Output From Alternative Query

If the documents that are retrieved do not meet the needs of the user and the user is unable to provide supplementary related keywords to search for, then the option of selecting additional search methods is of great significance. The thesaurus search and substring matching functions provide for greater versatility (see Figure 9). The thesaurus search option utilizes the keyword(s) that was/were entered by the user and then performs a search of the specific technical thesaurus data file for any related terms. These related terms are then

Netscape - [HyperPKM Indexing Gateway]

File Edit View Go Bookmarks Options Directory Help

Back Forward Home Reload Images Open Print Find Stop

HyperPKM Indexing Gateway

Type in the keyword or several keywords connected with "and" and "or".
Example: "picture and binary".

PPBS

You can restrict searches to documents that have changed during the last x days
(leave the other fields empty to get all documents changed during that time):

Number of days:

Turn on use of thesaurus to extend a search to all synonyms of a term, turn on
substring matching to extend searches to words which contain the given term as a
substring:

☒ Use thesaurus ☐ Substring matching

Limits search to a subtree of the document hierarchy:

Pre-Concept Exploration [Acquisition/Phase.0]

Start search:

Reset to default values:

Figure 9. Selection of the Thesaurus Search Feature

matched against the site index for any additional documents that are contained in the selected document hierarchy.

For this project, a specific technical thesaurus was developed utilizing a DOD acquisition manual and supporting documents. Numerous keywords were evaluated for corresponding word meanings and relevant relationships to other keywords. As in the case of the keyword "PPBS," a total of 12 additional keywords were found to be relevant. Three of these additional keywords (planning, programming and budgeting) would most likely be known to the user and probably searched for. The remaining nine keywords are most likely unknown to the user, in the context specified, and therefore, would present the user with additional information relating to the query. Figures 10 and 11 depict the documents that are generated as a result of searching for "PPBS" with the thesaurus function selected. As a result of this type of search, a total of 45 documents were found with 23 of them containing two or more distinct and related keywords. With the thesaurus feature, a search is conducted by topic and not by query. Hence, documents will be retrieved whether or not they contain the keyword(s) that was/were being searched for (see Figure 11).

Once a document is found to meet the users' needs, the selection of the documents title, which is a hypertext link, will take the user to the selected document (see Figure 12).

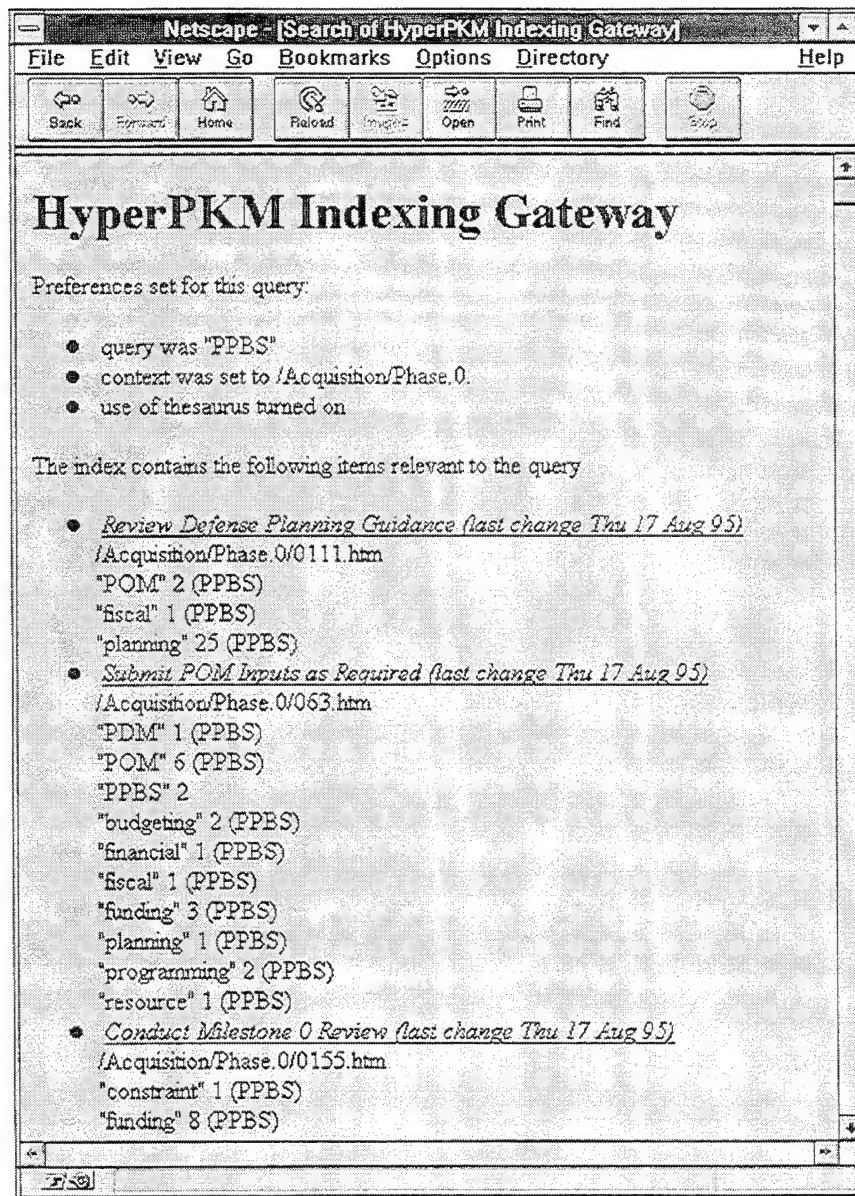


Figure 10. Search With Thesaurus Function Enabled

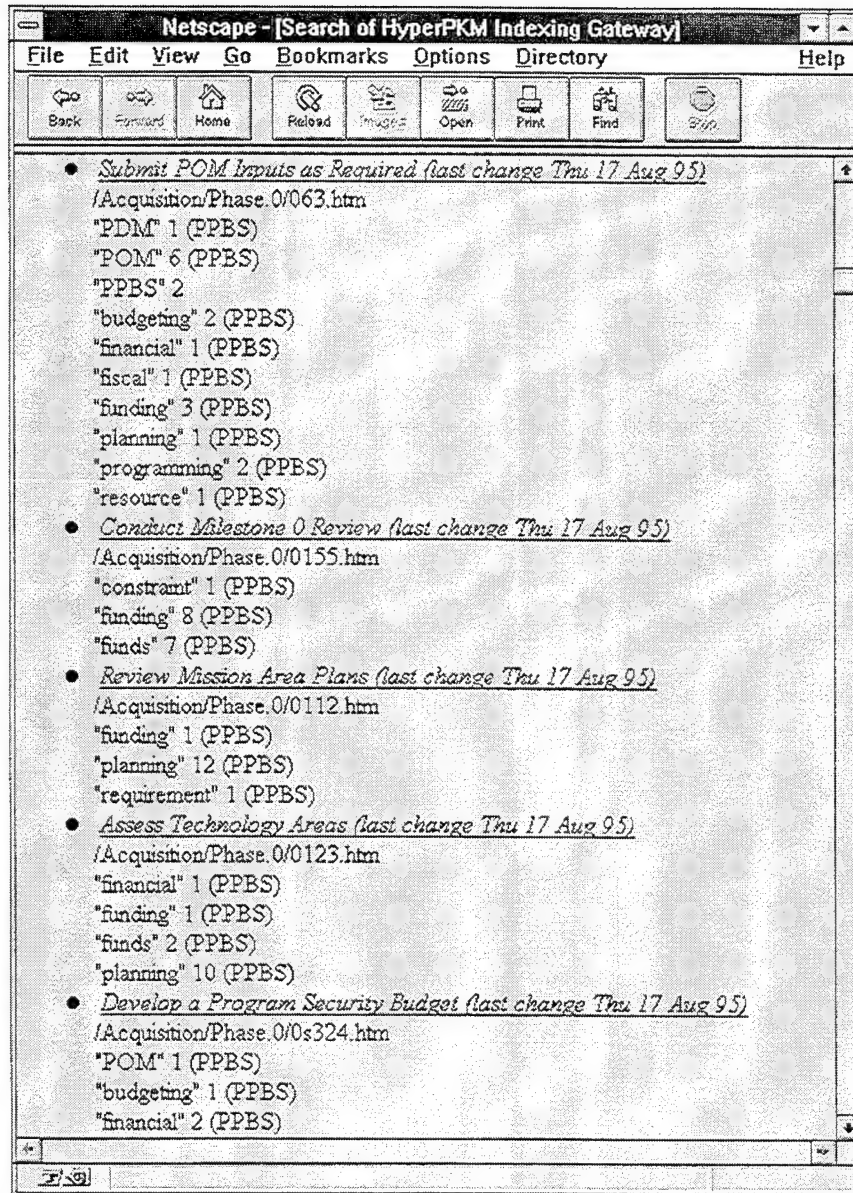


Figure 11. Thesaurus Search By Topic Area

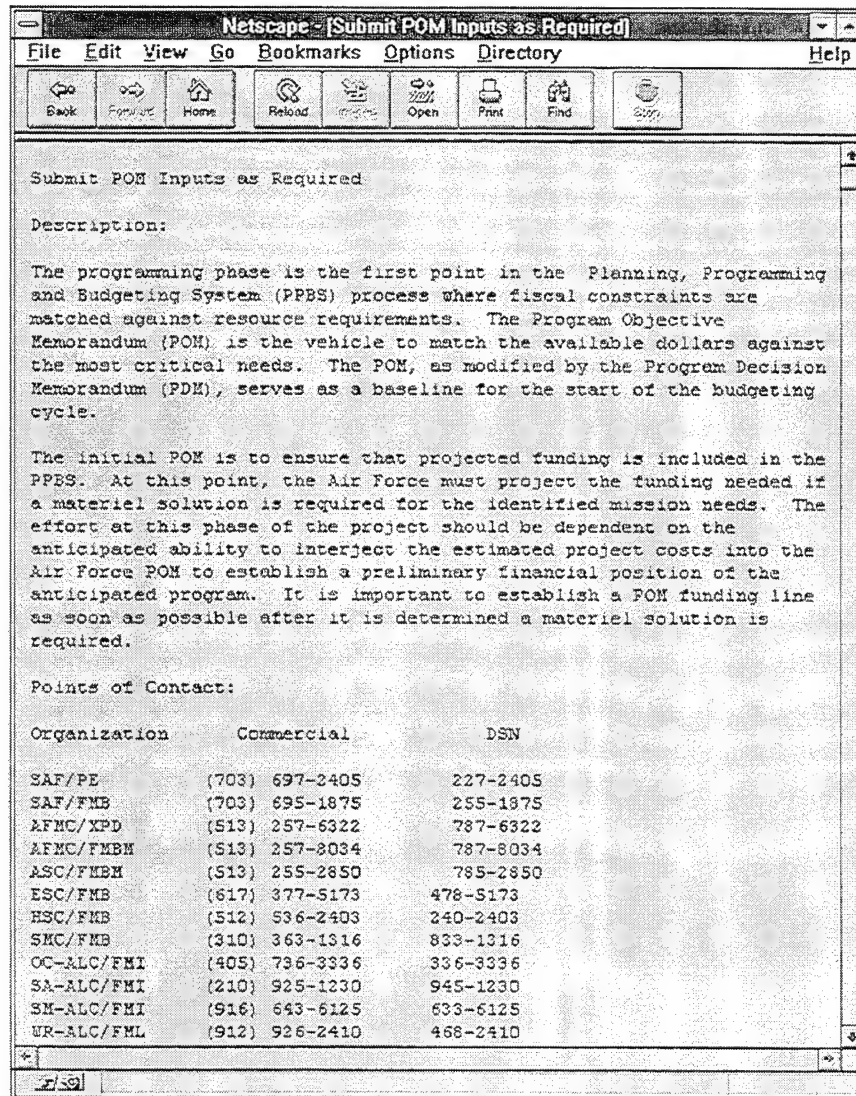


Figure 12. Document Retrieved Via Hypertext Link

VII. RECOMMENDATIONS AND CONCLUSIONS

A. DEVELOPER REQUIREMENTS

To continue development work with REMAP GBU, a developer should have a number of basic requisite skills. First, a good working knowledge of the UNIX operating system is required. A general knowledge of the Andrew Toolkit is also necessary so that the programmer understands the tools available and the object oriented environment which it provides. Additionally, a thorough knowledge of the C programming language is essential. Arrays, pointers and structures are used extensively in the application environment. The most invaluable knowledge was gained of the application environment by numerous hours of experimenting with the application and exploring the previous versions of the code.

Through the use of the PERL scripting language, the HyperPKM Indexing Gateway can be modified to provide greater search functionality for the user. Not only can the usefulness of the current search operators (boolean, thesauri, substring matching) be further enhanced, the ability to conduct proximity, wild card and conceptual searches can be implemented.

B. HELPFUL REFERENCES

The only comprehensive reference book available for the Andrew Toolkit is Nathaniel Borenstein's book *Multimedia Applications Development with the Andrew Toolkit*. This book is written for the advanced programmer with a solid base of C programming knowledge. The most helpful information concerning the Andrew Toolkit and the REMAP GBU environment can be obtained from the ConceptBase design team at the University of Aachen, in Aachen Germany. Throughout this project an extensive e-mail exchange was conducted with the team.

C. RECOMMENDATIONS

The Search Related Documents method could be improved to provide a better user interface. An improved interface might provide a dialog box to query the user as to what terms to search for, the appropriate boolean connectors to use, whether substring matching

or thesaurus searching is desired, etc. prior to passing those arguments to the search mechanism. This improvement would provide a cleaner interface with a mechanism to link and or save to the ConceptBase any additional information which help document the current design project.

D. FUTURE WORK

The HyperPKM model can be further modified to allow for "Smart Searches". That is, the ability to perform thematic content-oriented searches of documents contained within a host server. In utilizing thematic searches, an analysis of the document content, instead of word frequency, is conducted. This concept allows for documents to be retrieved that do not even contain any of the keywords queried or references to a thesaurus data file. Word connotations are understood by the indexing and search engine. The differences between "Computer monitor," "Health monitor" and "System monitor" are all understood. The "theme" of a document is comprehended. A document may contain discussions pertaining to "refresh rate," "screen size" or brand name, and would still be retrieved even if the term "computer monitor" was not in the document itself [Ref. 19].

A program such as Oracle'sTM TextServer 3 [Ref. 19] with the use of Oracle's database, can be incorporated into the HyperPKM model, thus, allowing the expansion of the search and retrieval capabilities of the HyperPKM search engine. Such an implementation is feasible in the context of the practicality and functionality of the HyperPKM model.

LIST OF REFERENCES

1. Blattner, Meera, and Danneberg, Roger, *Multimedia Interface Design*, New York: Addison-Wesley Publishing Company, p. xix , 1992.
2. Borenstein, Nathaniel, *Multimedia Applications Development with the Andrew Toolkit*, Englewood Cliffs: Prentice Hall, Inc., 1990 .
3. Ramesh, Balasubramaniam, and Dhar, Vasant, *Supporting Systems Development By Capturing Deliberations During Requirements Engineering*, *IEEE Expert*, Vol. 18:6, pp. 498- 510, 6 June 1992.
4. Ramesh, Balasubramaniam, and Dhar, Vasant, *Representing and Maintaining Process Knowledge for Large-Scale Systems Development*, *IEEE Computer Society Press*, pp. 54-59, April 1994.
5. Conklin, Jeff, and Begeman, Michael, *gIBIS: A Hypertext Tool for Exploratory Policy Discussion*, *ACM Transactions on Office Systems*, October 1988.
6. Rittel, Horst, *Dilemmas in a General Theory of Planning*, *Policy Sciences*, Vol. 4, 1973.
7. Jarke, Matthias, *ConceptBase V3.2 User Manual*, University of Aachen, p. 1, 1993.
8. Borenstein, Nathaniel, *Multimedia Applications Development with the Andrew Toolkit*, Englewood Cliffs: Prentice Hall, Inc., p. 13, 1990.
9. Palay, Andrew and others, *The Andrew Toolkit: An Overview*, paper presented at the UESNIX Association Winter Conference in Dallas, Texas, p.7, February 1988.
10. Beck, Howard W., Amir M. Mobini, and Viswanath Kadambari, *A Word is Worth 1000 Pictures: Natural Language Access to Digital Libraries*, <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/beck/beckmain.html>.

11. De Bra, P.M.E., and R.D.J. Post, *Searching for Arbitrary Information in the WWW: The Fish-Search for Mosaic*, <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/debra/article.html>.
12. Neuss, Christian, and Stefanie Höfling, *Lost in Hyperspace? Free Text Searches in the Web*, <http://www.igd.fhg.de/~neuss/w4-main.html>.
13. Levenshtein, V.I., *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*, Soviet Physics Doklady, Vol. 10, 1965.
14. Chen, Hsinchun, Bruce Schatz, Joanne Martinez, and Tobun Dorbin Ng, *Automatic Thesaurus Generation for FlyBase*, http://ai.bpa.arizona.edu/papers/sigir93/listoffigures3_2.html.
15. Chen, Hsinchun, Bruce Schatz, Tak Yim, and David Fye, *Automatic Thesaurus Generation for an Electronic Community System*, http://ai.bpa.arizona.edu/papers/worm94/listoffigures3_2.html.
16. Chen, Hsinchun, and Kevin J. Lynch, *Automatic Construction of Networks of Concepts Characterizing Document Databases*, http://ai.bpa.arizona.edu/papers/ieee91/listoftables3_2.html.
17. Chen, H., K. Basu, and T. Ng. *An Algorithmic Approach to Concept Exploration in a Large Knowledge Network (Automatic Thesaurus Consultation): Symbolic Branch-and-Bound Search vs. Connectionist Hopfield Net Activation*, http://ai.bpa.arizona.edu/papers/snnn92/listoffigures3_2.html.
18. Grefenstette, Gregory, *Automatic Thesaurus Generation from Raw Text using Knowledge-Poor Techniques*, <http://www.xerox.fr/grenoble/mltt/reports/home.html>.
19. *Information Highway or Information Ocean? Oracle Delivers Software to Control the Flood of Text-Based Information*, <http://www.oracle.com/info/news/textserver3.html>.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
Cameron Station
Alexandria, VA 22304-6145
2. Library, Code 013 2
Naval Postgraduate School
Monterey, CA 93943-5101
3. Prof. Balasubramaniam Ramesh (Code SM/Ra) 5
Naval Postgraduate School
Monterey, CA 93943-5103
4. Prof. Suresh Sridhar (Code SM/Sr) 1
Naval Postgraduate School
Monterey, CA 93943-5103
5. Christopher L. Vance 2
c/o Mr. & Mrs. A.B. Vance
917 Dorsett Way
New Bern, NC 28562
6. Kevin P. Sudhoff 2
c/o CDR & Mrs. Herb Sudhoff
Route #2
Sparta, TN 38583